T/CCUA

中国计算机用户协会 发布

2021-xx-xx实施

2021-xx-xx发布

基于微服务的云敏捷应用开发与集成要求

Requirements of cloud agile application development and integration based on microservice architecture

（征求意见稿）

（在提交反馈意见时，请将您了解的相关专利连同支持性文件一并附上）

本稿完成日期：2020-12-31

T/CCUA xxxx—2021

团体标准

ICS 35.080

CCS L77

目 次

[前 言 2](#_Toc61621227)

[1 范围 1](#_Toc61621228)

[2 规范性引用文件 1](#_Toc61621229)

[3 术语和定义 1](#_Toc61621230)

[4 缩略语 2](#_Toc61621231)

[5 云敏捷应用开发要求 2](#_Toc61621232)

[5.1 云敏捷应用开发概述 2](#_Toc61621233)

[5.2 云敏捷应用总体设计原则 2](#_Toc61621234)

[5.3 云敏捷应用架构选型推荐 3](#_Toc61621235)

[5.4 云敏捷应用接口开发要求 4](#_Toc61621236)

[5.5 云敏捷应用安全开发要求 5](#_Toc61621237)

[6 云敏捷应用集成要求 6](#_Toc61621238)

[6.1 接口集成 6](#_Toc61621239)

[6.2 数据集成 7](#_Toc61621240)

[6.3 客户端集成 7](#_Toc61621241)

[6.4 外部集成 7](#_Toc61621242)

[附录A 8](#_Toc61621243)

[(规范性） 8](#_Toc61621244)

[公司案例说明 8](#_Toc61621245)

[A.1 整体架构设计 8](#_Toc61621246)

[A.2 接口开发实践 9](#_Toc61621247)

[A.3 安全交互实践 11](#_Toc61621248)

[A.4 服务集成要求实践 12](#_Toc61621249)

前 言

本文件按照 GB/T 1.1-2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

本文件由中国计算机用户协会提出并归口。

本文件起草单位：中国计算机用户协会石油和化工信息技术应用分会、中国海洋石油集团有限公司、苏州博纳讯动软件有限公司、北京快益通科技有限公司、中国电信集团系统集成有限责任公司、新华三技术有限公司。

本文件主要起草人：李志刚、周斌、刘劲、冯雪、汪岳、宋歌、董志强、王连庆、郭浩、谢正华、童海帆、黄大治、李宁、杨烨、金多。

基于微服务的云敏捷应用开发与集成要求

1. 范围

本文件规定了微服务架构软件研发过程中应遵循的设计原则、开发与集成的要求。

本文件适用于企业采用微服务架构的业务系统进行微服务设计、敏捷应用设计的开发和集成场景。

1. 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件,仅该日期的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 36327-2018 信息技术 云计算 平台即服务(PaaS)应用程序管理要求

1. 术语和定义

下列术语和定义适用于本文件。

3.1

应用编程接口 application programming interfaces

面向程序的平台或服务的对外接口。

3.2

表征状态传输 representational state transfer

含状态传输的 Web 服务，一种网络应用程序的设计风格和开发方式。

3.3

跨域资源共享 cross origin resource sharing

一种允许当前域的资源被其他域的脚本请求访问的机制，通常由于同域安全策略浏览器会禁止这种跨域请求。

3.4

统一资源标识符 uniform resource identifier

用于标识某一互联网资源名称的字符串，该种标识允许用户对任何的资源通过特定的协议进行交互操作。

3.5

云敏捷 cloud smart

云敏捷指基于容器技术、DevOps和微服务等构建应用系统的模式。

3.6

超媒体 hypermedia

超媒体是超级媒体的缩写，一种采用非线性网状结构对块状多媒体信息（包括文本、图像、视频等）进行组织和管理的技术。

1. 缩略语

下列缩略语适用于本文件。

HTTP：超文本传输协议（HyperText Transfer Protocol）

API：应用编程接口(Application Programming Interfaces)

CORS：跨域资源共享(Cross Origin Resource Sharing)

URI：统一资源标识符(Uniform Resource Identifier)

DDD：领域驱动设计(Domain Driven Design)

EDA：事件驱动的架构(Event-Driven Architecture)

DNS：域名系统(Domain Name System)

TLS：传输层安全协议(Transport Layer Security)

CA：证书授权中心(Certificate Authority)

REST：表述性状态传递(Representational State Transfer)

SDK：软件开发工具包（Software Development Kit）

HATEOAS：超媒体应用状态引擎(Hypermedia As The Engine Of Application State）

1. 云敏捷应用开发要求
   1. 云敏捷应用开发概述

云敏捷应用在开发过程中，从生命周期维度，应遵循总体设计原则、架构选型推荐、应用接口设计要求、安全交互要求的内容，以原则、要求为执行准则，完成云端敏捷应用的开发。

* 1. 云敏捷应用总体设计原则

云端敏捷应用总体设计应遵循以下原则：

1. 领域驱动设计原则

一个团队负责一个领域业务功能的全部设计，团队拥有全系列的开发设计人员，包括用户界面、业务逻辑和持久化存储的开发设计技能。

1. 单一职责原则

每个服务包含一个独立的全部功能。

1. 接口发布原则

每个服务发布一个定义的接口，服务消费者只调用接口，对于被消费的服务没有任何运行依赖。

1. 服务细粒度原则

服务粒度应针对一个单一职责的业务能力进行封装。

1. 独立开发和演进原则

每个服务采用独立开发技术，松耦合，服务与服务之间采取与语言无关的API进行集成。

1. 独立部署与运行原则

每个服务能独立部署并运行在一个进程内。

1. 轻量级通信原则

服务通信使用轻量级的通信协议，例如：HTTP。

1. 支持多语言异构原则

每个服务自由选择开发语言、持久化存储、工具和方法。

1. 团队自治原则

团队对服务的生命周期负责，自决策自治理，无须统一的指挥中心。

* 1. 云敏捷应用架构选型推荐

5.3.1 分层架构

分层架构具有定义接口、业务逻辑和数据驱动切面划分特性，适用于以层次切分，业务逻辑不拆分而设计的应用系统。它分领域模型和业务逻辑，减少对基础设施、用户界面甚至应用层逻辑的依赖，将复杂的系统分为不同的层，每层都具有良好的内聚性。一个DDD中的分层架构（见图1），将领域层单独分离出来，负责表示业务状态及业务规则，形成对领域模型的汇聚，进而形成业务软件的核心。

表示层

Web服务接口

Web服务接口

Web应用服务

Web应用服务

领域层（包含业务逻辑）

服务1

资源库

服务2

基础设施层

数据库

图1 领域驱动设计分层架构

5.3.2 事件驱动架构

EDA是一种以事件为媒介，实现组件或服务之间松耦合的方式。解耦程度高，如果接口发生变化，代码不需变动，调用者和被调用者互相独立，通过中间消息队列通信。

按照GB/T 36327-2018确定的事件驱动有下列特征：

——生产者发生实时事件；

——推送通知；

——生产者发送即完成；

——消费者立即响应；

——事件与命令是有区别的。

一个事件驱动架构应包括的组件：

——Web服务；

——企业服务总线（针对复杂应用）；

——消息中间件；

——监控体系；

——异常处理的判断；

——配置和规则引擎。

5.3.3 六边形架构

六边形架构将上下或左右的分层架构变为内部和外部，六边形架构的结构（见图2）。六边形架构应用到领域驱动设计中，其内部代表了应用和领域层；外部代表应用的驱动逻辑、基础设施或其他应用。内部通过端口和外部系统通信，端口代表了一定协议，以API呈现。

一个典型的六边形架构应用有两个端口，一个端口对应用户接口层，用于应用控制，一个对应数据访问层，用于数据获取和持久化。每个端口都可对应几个适配器，该应用可被自动化测试，系统层面的回归测试，用户交互操作，远程HTTP调用，REST调用等。在数据方面，通过配置使用外部的数据库，从而实现应用和外部数据库的解耦。

采用六边形架构的系统中存在着很多端口和适配器的组合。端口表示一个软件系统的输入或输出，而适配器则是对每一个端口的访问方式。如要为系统增加新的访问方式，可为该访问方式添加一个相应的端口和适配器。

C

适配器E

适配器F

C

C

C

适配器A

应用程序

C

适配器B

领域模型

适配器G

C

适配器C

C

C

适配器D

适配器H

图2 六边形架构

* 1. 云敏捷应用接口开发要求

应用接口设计应遵守以下原则：

1. 简单易用

* 对API使用者友好，易于使用。
* 专注于URI和有效负载。

1. 一致性

* 确保API一致；例如：按照“/标识符/供应商/服务名称/版本号/资源”的构建方式确保API一致性。
* 以相同的方式实现搜索或过滤。

1. 灵活性

* 授权客户使用API。
* 尽量保持简单，例如：资源对象可采用英文名称或者唯一标识说明。

应用接口开发时遵守的要求，应不多于以下六种：

a) 客户端/服务器模式

* REST Web服务提供的统一接口将客户端与服务器分开。

b) 无状态

* 请求之间没有客户端上下文信息存储在服务器上。
* 每个请求均包含满足该请求所需的所有信息。
* 会话状态由客户端保持。

c) 缓存

* 代理可缓存响应。
* 响应隐式或显式定义是否可缓存，以防止客户端重用过期或不适当的数据。

d) 分层

* 可在客户端和服务器之间放置中间层，以提高可伸缩性，提供负载均衡等。

e) 统一接口

* 在请求中通过ID标识各个资源。
* 资源在概念上与返回给客户端的表示形式是分开的。
* 当客户端持有资源的表示形式时，它具有足够的信息来修改或删除资源。
* 每个消息都包含足以描述如何处理该消息的信息。

f) 超媒体作为应用程序状态引擎

* 资源表示形式与超链接一起提供，使客户更容易识别可能的操作。
* 客户端不预设任何特定操作用于特定资源。
  1. 云敏捷应用安全开发要求

5.5.1 数据传输安全

传输层安全至关重要。下列安全措施旨在帮助保护应用程序免受窃听和中间人攻击，并保护机密性，完整性和安全性。

1. 所有内容都应通过安全渠道提供：

* 应使用TLS 1.2及更高版本；
* 服务器应配置为确保完美前向保密性。

1. 应用程序的证书使用应遵守如下规则：

* 特定于应用程序的DNS域；
* 不是通配符证书；
* 始终提供所有需要的证书（即从证书到根CA的完整路径），以便客户端可完全验证信任链。

1. 不应在同一主机上混合使用加密和未加密的内容，因为通过HTTPS请求的任何资源都可能泄露会话标识符。

5.5.2 跨域资源共享安全

[CORS](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing)是用于配置跨域请求的标准。

API客户端发送特殊的HTTP请求标头分为两种：

——标头值标识请求客户端的方案、主机和端口；

——标头值在CORS预检请求中发送，以指示在客户端的实际请求中将使用HTTP方法。

安全规则分为两种：

——仅当应从其他域访问时，REST API才支持CORS；

——REST API应使用标头列出允许跨域访问其资源的资源列表。

5.5.3 密钥安全

密钥是与应用程序的每个合法用户或一组用户关联的密钥。

密钥可用于下列两种情况：

——识别：将接口的使用限制为合法用户；

——分析：跟踪使用情况，计算使用情况指标等。

密钥应具有下列属性：

——应满足一定长度和复杂度要求的随机字符串；

——应为唯一标识符；

——应随客户请求一起发送；

——在处理请求之前，应由服务器验证；

——应保持客户端，用户、设备或软件的唯一性。

密钥安全应遵守下列准则：

——不应将身份验证和授权仅基于密钥，例如密钥加动态口令验证；

——密钥不应在URL中传递，应通过标头或在客户端请求的主体内传递；

——密钥只能在安全通道上交换；

——密钥存储应以加密形式，例如基于Base64的加密；

——密钥读取只能由其所有者读取；

——密钥不应在用户界面中公开。

1. 云敏捷应用集成要求
   1. 接口集成

适用于业务逻辑集成，按照技术选型分为三种方式：服务调用，消息通知，服务总线。

——服务调用集成

* 服务调用协议选型：应依据应用实现框架进行选型，例如：HTTP协议、RPC协议。
* 规避系统耦合度：应通过标准数据传输格式缓解服务调用集成过程中耦合度，例如HATEOAS的要求。
* 系统集成可靠性：保证服务之间调用能够稳定可靠执行，应采用可靠机制，例如：重试机制，状态最终一致性。

——消息集成

* 消息组件：能提供消息传递功能的工具和框架，实现消息驱动的服务开发与交互能力，例如RabbitMQ，Kafka，RocketMQ。
* 消息获取方式：按照消费者的模式，以“推送”或者“拉取”的方式实现消费者。
* 消息识别：应定义生产者和消费者可识别的的唯一标识。

——服务总线集成

* 服务总线选型：例如Spring框架中的系统总线组件。
* 服务总线应支持多种不同集成断点。
* 服务总线SDK选型：根据不同开发语言，选择已经实现的SDK集成到系统中。例如Java、Python等。
* 消息分解与聚合：应满足消息分解并将消息分解发送后再进行组装。
  1. 数据集成

在微服务架构中，由于数据具备独立性，因此通过数据集成（共享数据库）不能满足微服务架构下的集成需求。同时针对一些遗留系统，通常采用“数据复制”的方式实现数据集成。数据集成过程中，应遵循如下要求：

——数据共享集成时存储及表现形式应保持稳定：例如数据表减少对表结构修改；

——数据复制集成时应通过多种方式保证数据一致性，例如通过最终一致性的方式实现数据批量复制或者基于事件机制实现数据复制；

——数据复制集成时需要对冗余数据进行处理。

* 1. 客户端集成

为满足客户端集成需要，面对多个微服务，应单独抽取一层作为客户端访问统一入口（例如API网关），实现后端多个服务的聚合，通过API网关实现客户端集成时，应满足以下要求：

——内部服务集成应统一服务间API接口，完成内部服务之间接口集成，实现不同服务之间可管控的访问；

——通过标准API的形式开放给外部合作伙伴，与客户端提供集成入口；

——集成过程通过标准协议（例如HTTP）对外提供接口，实现集成交付标准化。

* 1. 外部集成

外部集成主要针对业务系统需要集成第三方平台提供的服务场景。外部集成时采用回调的方式实现，在外部集成时应满足以下要求：

——不同协议SDK支持，服务与第三方应用集成时，需要与第三方应用暴露的接口集成，接口集成时需要根据应用接口通信协议，实现适配（例如HTTP协议适配），从而完成应用集成；

——数据适配与转换操作，应增加数据适配和转换层，生成符合本地服务的业务数据；

——业务回调接口，对于有服务交互的应用，需要满足交互反馈机制，例如可以通过回调接口方式，告知其他系统数据处理结果。

附录A

(规范性）

公司案例说明

A.1 整体架构设计

整体架构设计过程中遵循架构设计原则，按照角色、职责定义，场景分析，领域建模等方法论完成业务系统的整体建模。参考架构选型标准，最终完成整体架构设计（见图A.1）。

浏览器

Java应用

用户层

PC端

安桌

移动端

IOS

WEB

用户层

负载均衡

服务路由

业务服务层

业务类

XXX

管理

XXX

管理

XXX

管理

XXX

计划

XXX

支付

XXX

管理

XXX

管理

XXX

管理

。。。

公共类

XXX

中心

XXX

中心

XXX

中心

XXX

中心

XXX

中心

XXX

中心

XXX

中心

XXX

中心

。。。

搜索申请

搜索构建

基础服务层

机器人

深度学习

服务注册

服务发现

服务

治理

服务路由

服务配置

服务熔断

。。。

搜索

服务

智能

服务

计算

服务

锁

服务

离线计算

实时计算

锁管理

锁访问

XXX平台

日志采集

服务跟踪

日志

服务

数据库申请

数据库构建

数据

库

服务

分布管理

访问服务

FileNet

Oracle

HBase

FastDFS

MySQL

TSDB

容器申请

容器构建

容器

服务

弹性伸缩

限流保护

缓存申请

缓存构建

缓存

服务

镜像管理

镜像构建

镜像

服务

**硬件资源池**

监控通知

实时监控

智能运维

智能分析

自动恢复

测试自动化

敏捷交付

构建自动化

部署自动化

传输管理

版本管理

图A.1 系统整体设计示意图

系统在整体架构设计过程参考“云敏捷应用开发与集成要求”中总体设计原则和架构选型推荐内容，整体架构设计中采用分层方式，并以此为基础完成设计工作。

从分层以及各个层次承担的职责考虑，业务系统自底向上切分为五个层次：

1. 运行环境

以容器平台为基础，为基础组件服务（例如数据库）、基础业务服务（服务注册/发现），业务运行服务、用户控制台等提供运行应的计算资源，网络资源，存储资源等。

1. 基础支持服务

在分层架构中，提供系统需要的公共支持服务（例如日志服务，监控服务），基础支持服务在分层中，不承担具体的业务能力，只提供系统需要的非业务性功能说明。

1. 业务系统服务

在分层架构中，作为业务支撑的核心能力，业务系统层次切分为两个维度，一个是围绕业务场景本身，对业务内容进行处理工作（例如交易管理，报表管理）；另外一个是提供基础组件功能（例如统一认证鉴权中心）。

1. 统一服务接入

前端可视化与后端服务交互的链接层，由API网关充当此类角色，统一接入层负责后端服务聚合，协议转换，调用过程监控等。

1. 用户交互入口

指业务系统可视化操作的入口。

A.2 接口开发实践

A.2.1 API的URL通用格式

通用格式如下：

/api/domain\_namespace/service\_name/v1/{resource }?[params]

各参数/变量含义如下：

api——api接口的url模型头，进行负载均衡和路由；

domain\_namespace——服务域简码，可按照业务领域进行命名空间划分，比如生产服务、经营服务、营销服务等；

service\_name——具体服务名缩写，如库存管理服务、客户管理服务等；

v1——api版本，v1，v2，v1\_1，v2\_2；

resource——具体资源对象，如客户（customer）等；

params——参数部分，“&”分隔不同的参数。

A.2.2 HTTP请求头开发实践

a） Content type

* aplication/json：用于single和bulk时，用来表示请求数据为json格式。
* application/vnd.ms-excel：从excel格式的文件导入创建。
* application/merge-patch+json：更新操作时的格式信息。

b) Accept

* application/json：接受json格式的响应数据。
* application/vmd.ms-excel：接收excel格式的响应数据。
* application/pdf：接收pdf格式的响应数据。

c) Authorization

* Oauth2.0的access token（bearer token）。

d) If-Modified-Since

* 第一次请求该信息时服务器给的Last-modified时间戳。

e) Accept-Language（可选）

* 可接受的语言，例如： en-US表示美国英语。

A.2.3 HTTP请求URL参数

1. 对象过滤类：

* 说明：根据对象的属性确定范围，如精确到某一个对象或满足一定条件的对象集合；
* 关键字：{对象类型名}.{对象属性}=具体值或逗号分隔的值范围，如pod.id=1234、pod.ctime\_start=3040,3505，前者表示精确相等，后者表示取值范围。

1. 分页排序类：

* 说明：用于查询操作时对结果进行排序或分页控制；
* 关键字：offset、limit、sort，offset代表返回结果的起始页号，从0开始，limit表示每一页的条目数，最大100，sort代表基于那个参数进行排序，其值分为两个部分，第一个部分是属性，第二个是排序方式，如sort=name,asc，按照name的升序排列。

1. 结果内容范围：

* 说明：用于确定对返回结果的内容域的范围；
* 关键字：style、fields，style=compact表示对象的摘要格式，fields制定要返回的字段，如fields=name,id,location等。

1. 行为控制：

* 说明：用于控制服务端对某些异常或冲突的处理方式；
* 关键字：flow={sync，async}，conflict={skip，overlap}，比如遇到已经存在对象的重新创建，是跳过还是覆盖。

A.2.4 HTTP请求内容

A.2.4.1 Get请求

Get请求用于查询，查询参数放在url中，其内容域为空。

A.2.4.2 Post请求

Post请求用于创建新的对象（集），其内容域的格式由请求头中的Content type域指定内容区格式，如json或excel。当内容为json时，其格式如下：

{

content：[

{对象},

{}

]

}

A.2.4.3 Put请求

Put请求用于全量更新，其内容域的格式由请求头中的Content type域指定内容区格式，如json或excel。当内容为json时，其格式如下：

{

content：[

{对象},

{}

]

}

A.2.4.4 Delete请求

Delete请求用户删除对象（集），其内容域为空。

/api/cnooc/pres/v1/orders  
(标识)/（开发团体）/服务/版本/对象

具体实现代码如下所示：

|  |
| --- |
| @RestController  @Api(value = "用户服务")  @RequestMapping("/api/cloud/iams/v1")  public class UserController {  private final UserService userService;  private final PostAddressService postAddressService;  private final UserResourceAssembler userAssembler;  private final ResponseResourceAssembler responseResourceAssembler;  @Autowired  public UserController(UserService userService, PostAddressService postAddressService, UserResourceAssembler userAssembler,  ResponseResourceAssembler responseResourceAssembler) {  this.userService = userService;  this.postAddressService = postAddressService;  this.userAssembler = userAssembler;  this.responseResourceAssembler = responseResourceAssembler;  }  @ApiOperation(value = "删除用户", notes = "删除用户")  @AuditLog  @Authentication  @DeleteMapping(value = "/users/{id}")  public Response<Object> delete(@PathVariable("id") Long id) {  userService.remove(id);  return new Response<>(HttpStatus.NO\_CONTENT, "删除用户成功",  null);  }  } |

A.3 安全交互实践

应用开发过程中通过采取以下安全措施保护应用程序免受窃听，中间人攻击，并保护机密性，完整性和安全性：

——所有内容都应通过安全渠道提供

* 应使用/建议/接受TLS 1.2及更高版本。
* 服务器应配置为确保完美前向保密性。
* 包括静态内容，登录和注销页面。
* 这应避免的向量和关于混合内容的警告。

——应用程序的证书使用

* 特定于应用程序的DNS域。
* 不是通配符证书。
* 提供所有需要的证书，以便客户端可完全验证信任链。

——不能在同一主机上混合使用加密和未加密的内容，因为通过HTTPS请求的任何资源都可能会泄露会话标识符。

A.3.1 CORS安全实践

在项目开发过程中，为了保证跨域安全，对于GET，HEAD和POST以外的请求方法，CORS定义了预检请求交互。在客户端对跨域资源的实际请求之前，预检请求发生在符合CORS的用户代理和服务器之间的“幕后”。API客户端发送特殊的HTTP请求标头，例如Origin和Access-Control-Request-Method。

——Origin标头值标识请求客户端的方案/主机/端口。

——Access-Control-Request-Method标头值在CORS预检请求中发送，以指示在客户端的实际请求中将使用哪种HTTP方法。

A.3.2 密钥安全实践

API密钥是与您的应用程序的每个合法用户或一组用户关联的密钥。在密钥安全实践中通过以下方式保证密钥安全。

—— 使用的HTTP标头

* Authorization：理想情况下，您的API密钥应为[授权标头](#section-4.2)的参数之一。
* X-API-Key：另一种方法是使用客户HTTP标头。

—— 安全准则

* API密钥很敏感，但很容易丢失，被盗等（就像任何密码一样）。
* 不能将所有安全性都基于API密钥。
* API密钥可通过标头或在客户端请求的主体内传递，不可在URL中传递它们。
* API密钥可在安全通道上交换。
* API密钥可以加密形式存储。
* API密钥只能由其所有者读取。
* 只要认为API密钥可能已遭到破坏，可将其撤消并进行更新。
* API密钥不能在用户界面中公开。

根据上面规则生成的秘钥例子如下所示：

eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiJhNzc0NjVkY2JmMDA0NjZhOGQ2MzIzYmY1M2Y0MjBlZSIsInN1YiI6ImFkbWluIn0.CzgO5pqF-SFPYSuSunMmRUx\_4mBtYfq0rIW1bfczbG1xHQ1aFPS99eujBxbAAy0RXmT9k49gaGibcsstQ9w7wA

A.4 服务集成要求实践

业务系统集成时按照集成要求进行业务系统集成（见图A.2）。

内部应用

外部应用

**SaaS**

API网关

**服务接入**

鉴权中心

**服务安全**

安全策略

智能路由

**服务分配**

负载均衡

业务服务

**服务核心**

流程服务

系统服务

数据服务

事件服务

服务注册中心

**服务支撑**

服务配置仓库

熔断机制

服务消息总线

中间件/开源架构

**架构管理**

图A.2 系统集成示意图

系统在进行系统集成时，遵循云敏捷应用集成要求中提到的“接口集成”、 “外部集成”内容。具体集成过程如下所示：

——接口集成示例

系统采用微服务、分层架构。服务与服务之间集成时，参考“服务调用”要求，基于HTTP协议进行服务直接的调用。整体而言接口集成拆分为两个步骤：SDK引入，编写远程调用模板。

* SDK引入

<dependency>  
 <groupId>org.apache.httpcomponents</groupId>  
 <artifactId>httpclient</artifactId>  
 <version>${httpclient.version}</version>  
</dependency>

* 编写远程调用模板

public void logAfterThrowing(JoinPoint joinPoint, Throwable e)throws Throwable {  
 Long time = System.*currentTimeMillis*() - currentTime.get();  
 currentTime.remove();  
 HttpServletRequest request = RequestHolder.*getHttpServletRequest*();  
 *//构建Log对象* JSONObject jsonObject = buildLog("ERROR", time, getUsername(), StringUtils.*getBrowser*(request), StringUtils.*getIp*(request), (ProceedingJoinPoint) joinPoint);  
 jsonObject.put("exceptionDetail", ThrowableUtil.*getStackTrace*(e).getBytes());

//基于HTTP协议调用远程接口，实现服务集成  
 String result = HttpClientUtils.*sendPost*(LogProperties.*saveOperationLogUrl*, jsonObject.toJSONString());  
 *log*.info(result);  
}

通过以上方式通过调用第三方服务的HTTP接口实现服务与服务之间的调用集成。

——外部集成要求示例

外部集成时参考外部集成要求，各个服务进行集成时，分为两个步骤：第一步，引入统一集成 组件的SDK；第二步，各个业务系统基于接口的方式直接注册到API网关，实现多场景下的业务服务集成。

* 引入统一集成组件的SDK

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

<version>2.1.2.RELEASE</version>

</dependency>

* 注册到API网关

port: 8081

spring:

application:

name: gateway-service

cloud:

gateway:

discovery:

locator:

enabled: false

#开启小写验证，默认feign根据服务名查找都是用的全大写

lowerCaseServiceId: true

routes:

- id: client-test

uri: lb://CLIENT-TEST

predicates:

- Path=/testclient/\*\*

filters:

- StripPrefix=1

- id: service-feign

uri: lb://FEIGN

predicates:

- Path=/service-feign/\*\*

filters:

- StripPrefix=1

eureka:

instance:

preferIpAddress: true

instance-id: ${spring.cloud.client.ip-address}:${server.port}

client:

service-url:

defaultZone: http://localhost:8761/eureka/

通过以上两个步骤，实现外部集成。